User-oriented Network Security Policy Specification

Fulvio Valenza[1,2*], and Antonio Lioy[1]
[1]Politecnico di Torino, DAUIN, corso duca degli Abruzzi 24, Turin, Italy
[2]CNR-IEIIT, corso duca degli Abruzzi 24, Turin, Italy
{fulvio.valenza, antonio.lioy}@polito.it

Abstract

The configuration and management of security controls and applications is complex and not well understood by the majority of end-users (i.e. it typically requires specific skills). The security policy language simplifies this task and reduces the number of errors and anomalies. This paper proposes the specification of the two mechanisms for defining user's security policies, namely High-level Security Policy Language (HSPL) and Medium-level Security Policy Language (MSPL). HSPL is suitable for expressing the protection requirements of typical non-technical users, while MSPL is a lower-level abstraction useful for expressing specific configurations of security controls in a generic format (as such it is more appealing for technical users).

Keywords: network security policy, security requirement, policy refinement

1 Introduction

Nowadays the common approach to protect personal devices from Internet threats relies on the installation of appropriate security controls (e.g. firewall, VPN concentrator, etc.).

To achieve this goal, typically it is required a deep knowledge on how each security control should be configured, which generally involves in setting up several security applications of different vendors and different security functions (or capabilities), like packet filtering, VPN gateway, parental control, etc.. In general, for non-technical users and occasionally for administrators, this may turn out a difficult hurdle to overcome[21].

In order to simplify the configuration of security controls, we propose the definition of two user-oriented network security policy languages. The main difference between these languages is that they are designed and oriented to be adopted by final users with different skills and experience in programming security controls. The former, i.e. High Security Policy Language (HSPL) is oriented to non-technical users, e.g. business men or parents. The latter, i.e. Medium Security Policy Language (MSPL) is suitable for expert users (e.g. network administrators, IT operators).

The idea of HSPL is to define a policy language by using high-level security requirements, with the following requirements:

- **simplicity**: the definition of a security policy must be intuitive; for this, the user must be assisted by using predefined statements (e.g. "block Internet traffic") and auto completion techniques;
- **flexibility**: to define every type of security policy, also supporting specific conditions (e.g. time constrains, content types, traffic types), for every security application;
- **extensibility**: the language must support future extensions, e.g. by introducing new policy types and specific conditions without changing the structure of HSPL.

Journal of Internet Services and Information Security (JISIS), volume: 8, number: 2 (May 2018), pp. 33-47
*Corresponding author: Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi, 24, 10129 Torino , Tel: +39-(0)11-090-7192

On the contrary, MSPL should satisfy the following requirements:

- **Abstraction**: the language must contain abstract security-related configurations, independent from a given vendor or product specific representation and storage. The reason for this requirement is that configuration semantics are independent from the actual representation. In fact, the same configuration settings can be represented and enforced in different security controls.
- **Diversity**: it must support the description of configurations for a variety of security functions (confidentiality, filtering, etc.). The configuration meta-model must furthermore support the configuration of such security capabilities, which follow different policies and concepts (e.g. communication protection, parental control), and are applied to different types of security controls.
- **Flexibility and extensibility**: it must be flexible and extensible enough to support the introduction of new security controls.
- **Continuity**: it must ensure the continuity of the policy chain, starting from HSPL down to the security control settings. This is useful to tracking which policy is actually enforced and which user is associated to it.

The rest of the paper is organized as follows: first, an outline is provided to describe the essentials of the proposed paradigm (Section 2); then, HSPL and MSPL languages are presented (Section 3 and Section 4); finally, we discuss the related works (Section 5) and conclusions (Section 6).

2 Policy Language Abstraction

Our approach is based on three policy abstraction layers: High-level Security Policy Language (HSPL), Medium-level Security Policy Language (MSPL) and the low-level configurations, needed for setting up a specific security control.

HSPL has been designed to express security requirements by means of "*subject-verb-object-parameters*" sentences (as an authorization language [10]). A security policy is expressed as a set of sentences close to natural language, e.g. "do not download spam emails", "do not access social network", "block Internet traffic in the night".

The terms of a sentence (subject, object, etc.) are chosen by the user from a predefined set as different lists, i.e. a list for each element (e.g. action, subject). This approach avoid the user to learn new languages (it is transparent for users) and makes it possible to map each element of a terms to the related HSPL component. It is clear that, users is able to personalize some elements of a sentence, for example to define timing constraints, particular URL, etc.

MSPL, instead, has been defined to abstract the configuration languages with a vendor and control-independent format, which is organized by capabilities (e.g. a firewall from a particular vendor may implement packet filtering functionality in its capability set, while another vendor may implement also an deep packet inspection capability in her firewall). Unfortunately, defining this abstraction is not trivial because each security control has a specific syntax. Therefore the mapping can be unmanageable in a generic syntax and MSPL is organized by security capabilities. A Capability is a basic feature offered by a security control (e.g. filtering, anti-spam, data protection, parental control). Therefore, MSPL is organized by a general model that defines the high-level concepts (policies, rules, conditions, actions, etc.) and a set of sub-models to capture the semantics specific concepts as attributes, condition types, methods, etc.

On the other hand, a control-specific configuration language depends on formats and features available at the actual security control: each security controls are their own configuration languages. This means that two of the three abstraction layers, HSPL and MSPL, are under our control.

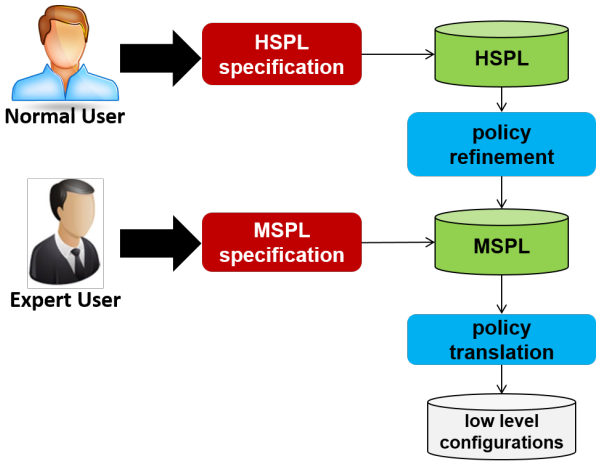

Figure 1: From network security policy languages to security control configurations.

As shown in Figure 1, there are two processes that can enable the auto-generation of security controls configuration: (1) the refinement of HSPL into MSPL ("*policy refinement*" in figure); (2) the translation of MSPL policies into low-level configurations ("*policy translation*").

Policy refinement is a sophisticated process "to determine the resources needed to satisfy policy requirements, to translate high-level policies into operational policies that may be enforced by the system" [16, 4].

The translation of MSPL policies into control-configurations (i.e. "*policy translation*") involves only change of syntax, as MSPL has been designed to share the same semantics as the security controls. To support a wide set of low-level security controls, the translation must be designed to support different languages (e.g. netfilter/iptables or PF for a stateful firewall). Therefore there is a set of translation modules that takes as input MSPL statements and generates the specific configuration for each security control.

In the next sections, we present more in details the two policy languages, while the refinement of HSPL into MSPL and the translators from MSPL to low-level settings are out of the scope of this paper and will be discussed in another work.

3 High-level Security Policy

HSPL is composed by statements with the following structure:

```
[ sbj ] action obj [(field_type,val)...(field_type,val)]
```

Where:

- `sbj` is the user who needs to access or perform some operation on an object (e.g. employee, family member) and may be omitted if the policy is applied to the user that defines the HSPL;
- `action` is the operation performed on the object (e.g. protect, permit access, enable, authorize access, etc.);
- `obj` is the entity target of the action (e.g. a resource such as e-mail scanning, Internet traffic, P2P traffic, 4G services, etc.);

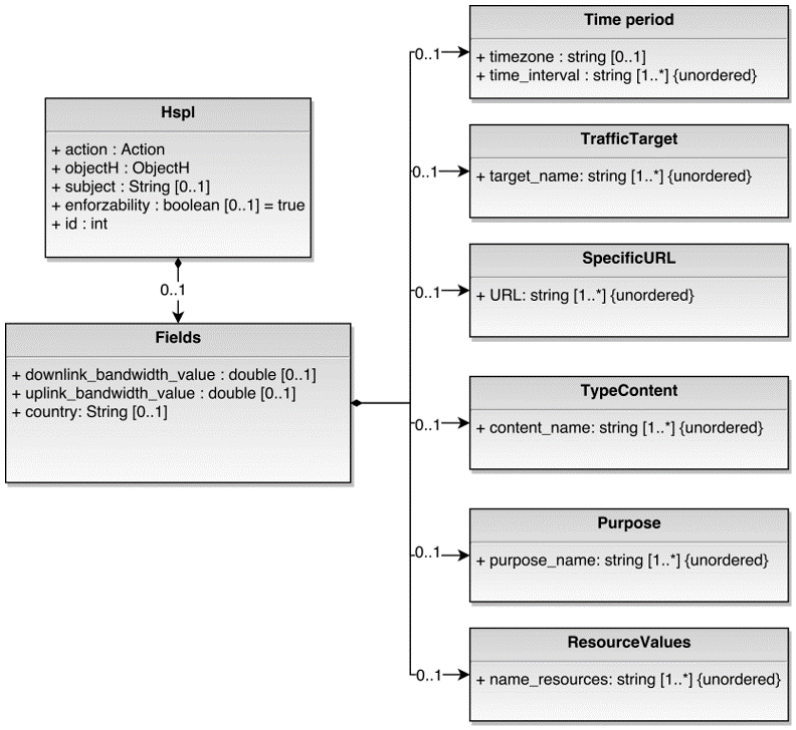

Figure 2: HSPL structure.

- `(field_type,value)` is an optional condition that add specific constraints to the action (e.g. time, content type, traffic type). The value part is a string with specific format depending on the field type.

Action Considering the main goal of HSPL, i.e. expressing concepts related to end-point protection, a set of predefined actions is proposed. The following set of actions are:

- *is/are (not) authorized to access*: to explicitly allow (deny) a generic type of traffic (e.g. Internet), a specific type of traffic (e.g. VoIP, P2P, DNS traffic) or the traffic related to a particular resource (e.g. FTP service);
- *enable(s)*: to activate a particular type of control (e.g. antivirus, e-mail scanning, parental control) or feature (e.g. DDoS protection, logging);
- *remove(s)*: to intercept and block specific traffic (e.g. related to advertisement or tracking techniques);
- *reduce(s)*: to limit the usage of bandwidth;
- *check(s) over*: to analyse network vulnerabilities;
- *count(s)*: to trace and limit the number of connections (e.g. DNS packets);
- *protect(s) confidentiality and/or integrity*: to ensure confidentiality and/or integrity of a particular data flow.

Object The object represents the entity controlled by the action (e.g. particular type of traffic, particular resource). To facilitate the creation of a HSPL, a predefined set of objects is provided. In particular, the HSPL supports the following objects:

- *Internet/Intranet traffic*: the traffic involving IP addresses different from private addressing or IP addresses of the intranet;
- *DNS traffic*: respectively the traffic of DNS, the DNS requests or DNS responses;
- *VoIP traffic* the traffic regarding VoIP (e.g. SIP);
- *3G/4G traffic*: the traffic regarding 3G/4G;
- *public identity*: to masquerade the identity of the user;
- *resource "x"*: the traffic regarding a particular resource ("x"), better specified by using a field condition;
- *advanced/basic parental control*: parental control with advanced/basic features;
- *antivirus*: antivirus features;
- *logging*: to trace user's activities (e.g. visited web sites);
- *IDS*, *IPS*: intrusion detection and prevention features;
- *DDos attack*: protection against distributed DoS attacks;
- *email scanning*, *file scanning*: specific objects like e-mail messages or files;
- *tracking techniques*, *advertisement*: common techniques used to track users or advertisements;
- *bandwidth*: to control bandwidth (e.g. to reduce it);
- *connection*: to control the number of connections.

File conditions A set of optional parameters is available to better specify the scope of an object (e.g. to define that an action is applied only for a particular type of traffic). These parameters are implemented by using field conditions. A field is organized in two parts: `field_type` and `value`. The following fields and available values are identified:

- *time*: to restrict the application of a HSPL policy statement with timing conditions, e.g. date range or particular days. The value part of a time range contains starting date/time, ending date/time and a time zone (expressed by using GMT notation) following the format:

  $$\{\texttt{[y:m:d]h:m[:s]-[y:m:d]h:m[:s],GMT}\}$$

  It is possible to define more intervals by using a comma separation. To define particular days, the field supports the use of the following keywords: "`weekend`" and all days of week (e.g. "`monday`").

- *specific URL*: to restrict the policy considering a specific URL (e.g. www.twitter.com). In this case the value part contains URLs in the format $\{$`url1, ...urlN`$\}$.

- *type of content*: to restrict the policy considering specific contents. For example social networks (e.g. Facebook, Twitter), illegal content, gambling, explicit sexual content, etc. The adopted format is {`content1, ..., contentN`}.
- *traffic target*: to specify the traffic of a particular user, organization or company e.g. corporate traffic ({`corporate network`}), user1's traffic ({`user1`}). Obviously the keywords adopted for the target must be defined by the user (who creates the policy) for each organization;
- *purpose*: to specify the type of analysis, the supported types are malware detection, spam detection by using the format {`purpose1,..., proposeN`}.

Note that the set of possible values for action, object and field condition of HSPL is not limited to the one presented in this paper, because it can be further enriched in the future (e.g. "*5G traffic*" may be a possible object for covering also the recent innovation of 5G services).

3.1 Examples of HSPL

This section collects a set of possible HSPLs, providing their description with their syntactic representations.

Deny access to social network web site A security policy to block access to social network for Alice. The corresponding HSPL statement is:

```
Alice is not authorized to access Internet
    (type of content, {social network})
```

Permit Internet traffic for a specific time slot A common policy is to permit access to Internet (e.g. to visit social network websites) only on predefined time slots, e.g. during lunch time. The corresponding HSPL statement (for user "*Alice*") is:

```
Alice is authorized to access Internet traffic
   (time period, {12:30-13:30 Europe/Rome})
```

Encrypt traffic of corporate network Finally, the encryption of business information, trade secrets, etc. are quite important to avoid information disclosure:

```
Bob protects confidentiality intranet
 (traffic target, {marketing-subnet})
```

4 Medium-level Security Policy

The base idea of MSPL is to describe the configuration settings for a class of security controls.

More in details, MSPL is an abstract language with statements related to the typical actions performed by various security controls (e.g. matching patterns against packet headers, keeping track of connection status, identifying the MIME type of a payload), but expressed in a generic syntax.

As discussed in Section 3, HSPL is suitable for capturing the user requirements, however it cannot be directly implemented by security controls. This requires the definition of a medium-level language able to express the same information into operational policies by using a format suitable for configuring security controls. This type of language is an ordered sequence of actions (e.g. permit and deny for a

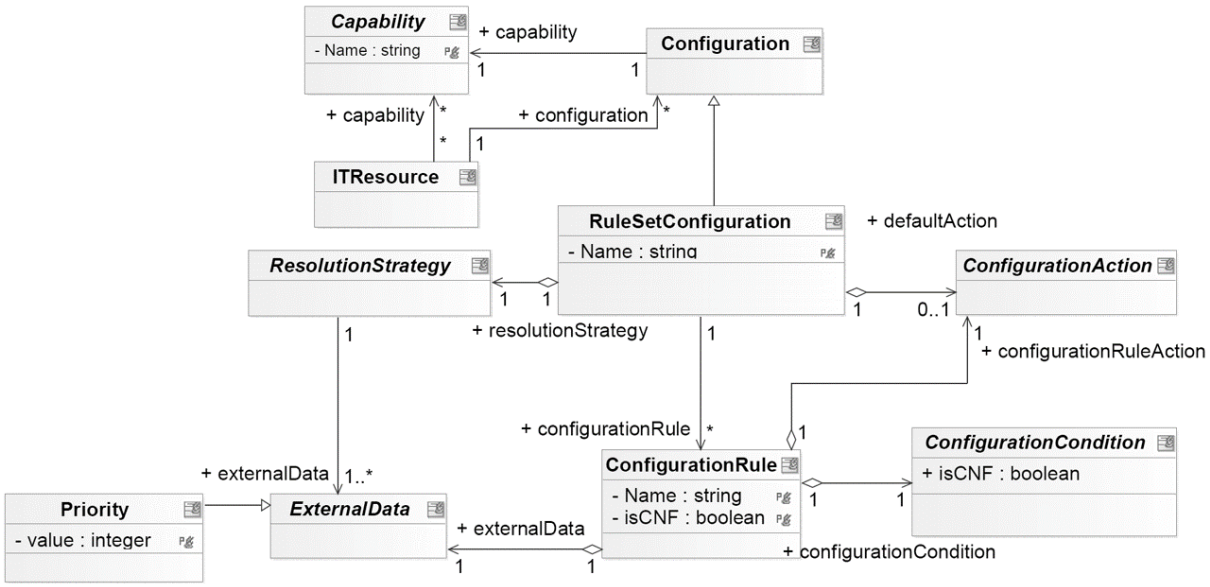

Figure 3: General configuration meta-model.

firewall) related to matching packets or payloads. The proposed model is defined by using the Unified Modeling Language (UML). Figure 3 sketches the elements of the configuration meta-model. First of all, we concentrate on core elements which are `ITResource`, `Capability` and `Configuration`.

An `ITResource`, is the central concept and represents a piece of software that implement a security control. For each `ITResource`, we can collect the capabilities being supported by this software functions In addition, zero or more configurations should be assigned for each capability of the `ITResource`. This assignment is done by the relation `configuration`. The association class with the same name allows the qualification of different types of configurations for an `ITResource` instance in the model. It can be used to represent the current configuration of an `ITResource` and the golden configuration as a result of the policy refinement process, at the same time.

A `Capability` represent any kind of network and security functionality that can be provided, e.g. filtering, data protection, anti-spam. In this work, mainly common security capabilities will be considered, (i.e. function that are typically supported by a class of software products).

A `Configuration` is an abstract configuration settings, which are independent of a given product or vendor. The `Configuration` class can be subclassed by configurations dedicated to deliver the different capabilities, e.g. class `FilteringConfiguration` (shown in Figure 4) to describe settings for packet filtering of a `Firewall` component. To ensure the identification of the affected infrastructure element for a given configuration, each `Configuration` is assigned by the relation `configuration` to exactly one `ITResource`.

A `RuleSetConfiguration` is a specialization class used to represent a configuration. The `RuleSetConfiguration` is the expected outcome of the policy refinement process, that is, a rule set is the representation of a MSPL policy for a security controls. Each `RuleSetConfiguration` consists of a set of `ConfigurationRule`.

A `ConfigurationRule` is enforced by `ConfigurationAction` (association `configuration-RuleAction`) and can use a set of `ConfigurationCondition`. With the association `defaultAction` a default `ConfigurationAction` is linked to the `RuleSetConfiguration` in order to specify what to do when no rule can be applied. If more than one `ConfigurationRule` can be applied, the `ResolutionStrategy` assigned to the `RuleSetConfiguration` via the association

`resolutionStrategy` is used to solve the conflict.

4.1 Filtering MSPL

This section provides a relevant example of configuration meta-model, that is the Filtering MSPL, whose details are shown in Figure 4.

The `RuleSetConfiguration` is a subclass of the `Configuration`. To describe a filtering configuration we specialize `ConfigurationAction` and `ConfigurationCondition`. The other classes belong to the general configuration meta-model presented in Figure 3. A filtering configuration can be represented as a set of `ConfigurationRules` that use conditions that are subclasses of the `FilteringConfigurationCondition` class and enforce (exactly) one action that is subclass of `FilteringAction`. The default action must be taken from the `FilteringAction` represented by the attribute value `FilteringActionType`, which usually allowed filtering in Allow and Deny; some devices also support the Reject action that discards the packet but also sends an ICMP notification to the sender. The resolution strategy for a filtering configuration may be one of the available strategies i.e. FMR, ALL, DTP, ATP, MSTP and LST. Filtering devices are divided according to the ISO/OSI stack at which they can work. Most of the firewalls are able to inspect packets at layer 3 and layer 4, some of them can inspect packets up to layer 7 (application layer firewalls). Application layer firewalls support the possibility of specifying conditions on protocol-specific fields, for instance, Squid, supports filtering on HTTP protocol headers. Recently, some products have been made available able to watch inside the content at the application layer (content filters). To represent this scenario, our configuration meta-model introduces two specialization classes of the `FilteringConfigurationCondition`, i.e. `PacketFilterCondition` and `ApplicationLayerCondition`. The former includes a number of attributes that are needed to select the packets to which to apply the action: the source and destination IP addresses and ports (`sourceAddress`, `sourcePort`, `destinationAddress`, `destinationPort`), the protocol type field in the IP header (`protocolType`), and the direction (`direction`, i.e. inbound or outbound). The latter includes the attributes typically related to the application layer, i.e. URL (`URL`) and particular methods (e.g. `httpMethod` for the HTTP protocol).

Moreover, filtering devices are categorized according to their ability to maintain state information [3]. Devices that do not maintain state information are named packet filters, while the others are named stateful filtering devices; in particular the functionality they implement is named stateful inspection if the analysis works at transport level or stateful protocol analysis if it works at application layer. Stateful devices keep track of each connection by examining certain values of TCP and other protocols headers and maintain a state table. State tables contain an entry for each of the observed (or to reduce memory allocation only the allowed) connections, usually represented by a five-tuple composed by the IP source and destination addresses, the source and destination port, and the protocol type. Stateful conditions always refer to some packet filter condition. Values in the state table are used to make decisions, e.g. allowing all the TCP packets related to an established connection, and sometimes implicitly, e.g. blocking packets that do not comply with the TCP protocol specification. Stateful inspection sometimes enforces a simple form of bandwidth control, that is, to specify the maximum number of connections allowed to a given destination, and limits the packets rate per destination address base or per port base. An improved type of application firewall is the application-proxy gateway, which enforces an access control policy using a proxy agent. Application-proxy gateway may keep track of authenticated users, to permit the specification of rules limiting the maximum number of allowed users, or the maximum number of connections on a per user base. To model stateful filtering, we introduced the abstract class `StatefulCondition`. Its subclasses permit to describe:

- `State`, to describe conditions on states, for instance, if the RELATED connections should be

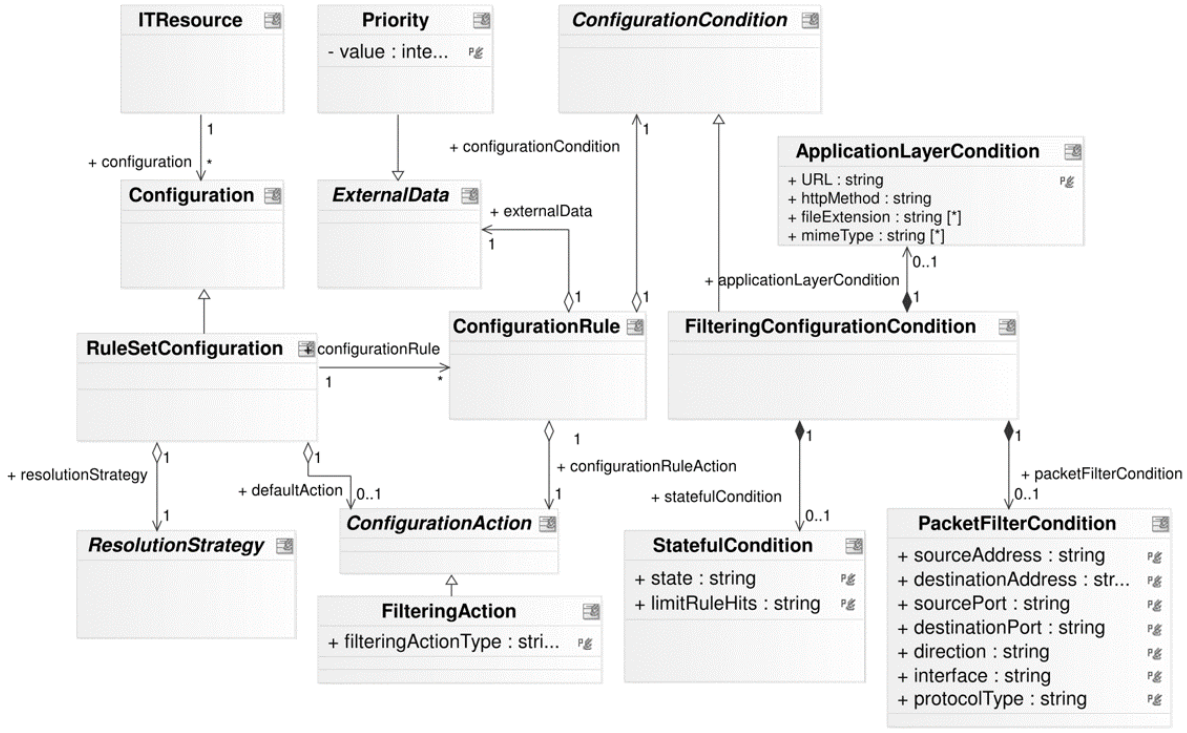

Figure 4: Filtering configuration.

allowed as well (as for FTP data and connection flows) or ESTABLISHED, used to permit the TCP traffic back for communications that are allowed in one sense and that correctly terminated the TCP three way handshake;

- `limitRuleHits`, allows to describe conditions on bandwidth and other counters.

4.1.1 Data protection configuration

This section provides a relevant example of configuration meta-model: the Data Protection MSPL Figure 5 and Figure 6)).

Data protection can be enforced in different ways: channel protection, static data protection and message protection [19, 20]. A secure channel is created agreeing on one or more symmetric cryptographic keys that are then used to cipher or authenticate data (using for instance a HMAC or keyed digest). A challenge is often performed in order to authenticate peers (nevertheless, the challenge is a feature of the authentication capability). Examples of protocols and standards able to create a secure channel are IPSec, WPA2, SSL/TLS, SSH. Additionally, there are techniques that can be applied to protect data to be transferred independently from the channel. This process is often named "message protection".

The configuration meta-model, describe in this work, must be able to describe every type of data protection method in an abstract way. The specialization model defined to this purpose is the `DataProtection` configuration meta-model (see Figure 5 and Figure 6). Similarly to filtering configuration, `RuleSetConfiguration` is a specialization of `Configuration` class. To describe data protection configurations it is necessary to subclass the `ConfigurationAction` (Figure 5) and the `ConfigurationCondition` (Figure 6).

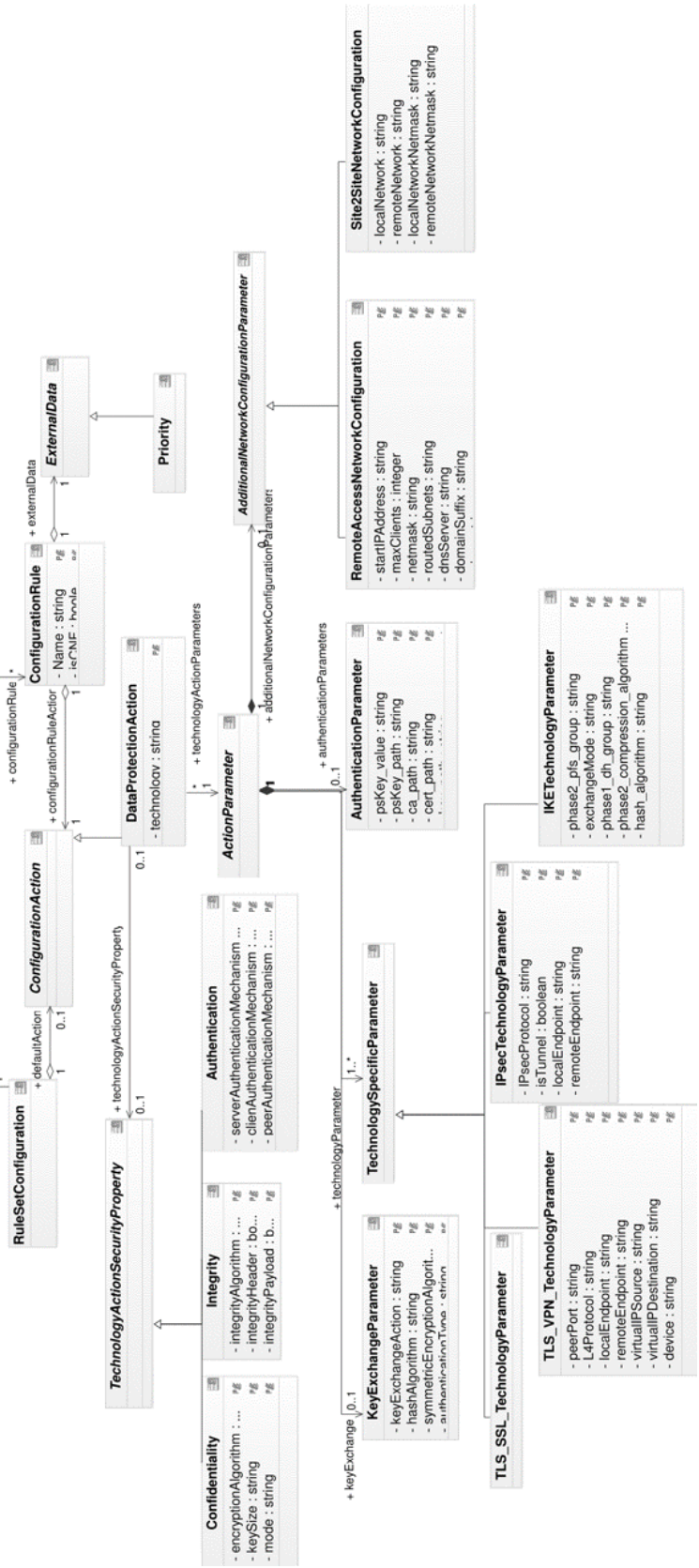

Figure 5: Data protection configuration - part 1

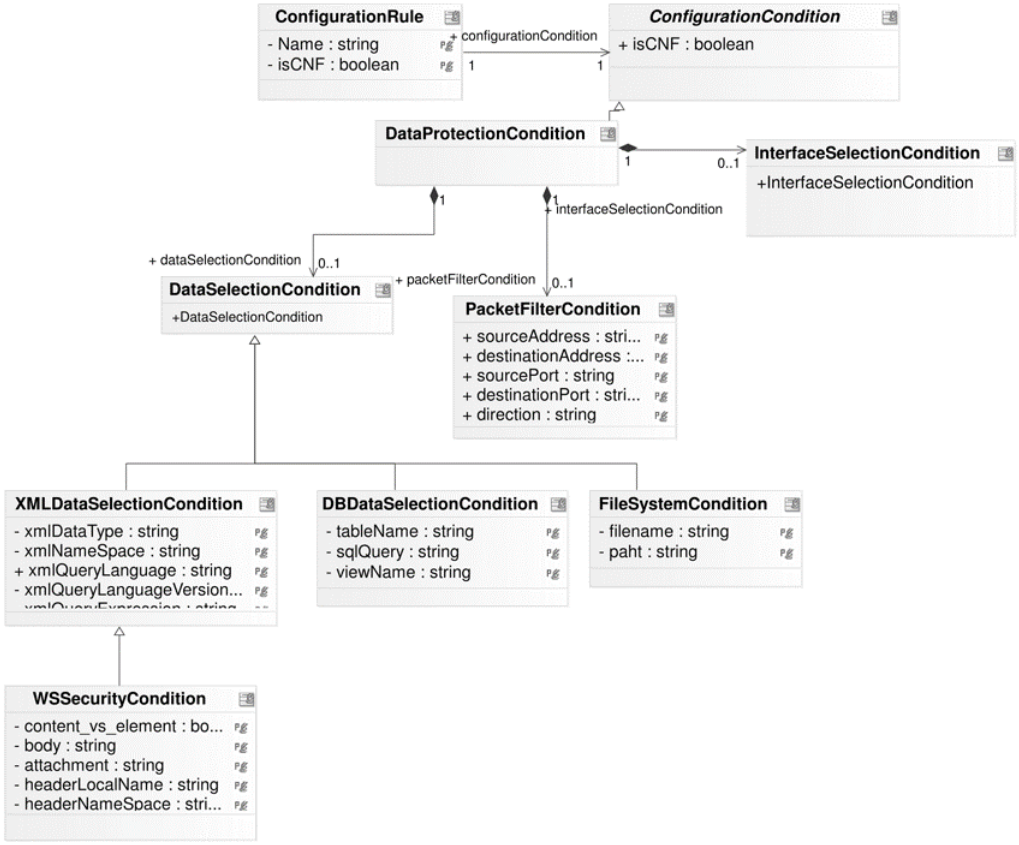

Figure 6: Data protection configuration - part 2

The `DataProtectionAction` specializes `ConfigurationAction` and contains the attribute `technology`, used to convey precise information about the technology to which the abstract configuration maps to. Every `DataProtectionAction` is associated to other technology specific information conveyed by the class `ActionParameter`. The subclasses of that class are used to better characterize the parameters to apply when the used technology is known. In particular, `KeyExchangeParameter` (by the association `keyExchange`) permits to specify the key exchange action, the authentication type, the algorithms for hash and symmetric encryption. The class `AuthenticationParameters` (by the association `authenticationParameters`) contains information on pre-shared keys (values and path), CA (path), certificate (path) and key (path), typically used for IPSec and SSL/TLS VPNs. The abstract class `TechnologySpecificParamters` contains the specific parameters for different data protection technologies.

This class is specialized as:

- `TLS_SSL_TechnologyParameter` that actually does not contain specific attributes;
- `TLS_VPN_TechnologyParameter` specifies attributes for configuring SSL/TLS-VPN (e.g. OpenVPN [1]). In particular the port to adopt (e.g. 1194/TCP), local and remote endpoint (expressed as IP addresses or hostnames), device (e.g. tun0), TLS mode (e.g. tls-server, tls-client);
- `IPsecTechnologyParameter` specifies attributes for configuring IPSec. In particular the

[1] `http://openvpn.net`

protocol (e.g. ESP, AH), tunnel or transport mode, local and remote endpoint (expressed as IP addresses or hostnames);

- `IKETechnologyParameter` applied only to IPSec configurations, specifies IKE attributes. In particular, exchange mode (e.g. main), hash algorithm (e.g. SHA1) and attributes for initialization phases.

The configuration of VPNs often requires information related to the network. By using the abstract class `AdditionalNetworkConfigurationParameters` (and the association `additional-NetworkConfigurationParameters`) two specialization are provided to describe remote access and site to site scenarios. The class `RemoteAccessNetworkConfiguration` specifies the attributes to configure remote access scenario, including the IP addresses for clients, the routing information to reach other networks, DNS servers, etc.. Similarly, the class `Site2SiteNetworkConfiguration` specifies the network information on local and remote network, including IP addresses and netmasks.

The class `DataProtectionAction` also has a set of security properties (by using the association `technology/ActionSecurityProperties`) related to integrity, confidentiality and authentication. The class `Technology/ActionSecurityProperties` contains the subclasses of security properties, i.e. `Integrity`, `Confidentiality` and `Authentication`. The first class contains the details on integrity algorithm (e.g. HMAC-SHA1), and boolean value for integrity of header and payload (e.g. true when a property is required). The confidentiality class has attributes to specify encryption algorithm (e.g. AES), the mode (e.g. CBD) and the key length in bit (e.g. 256). The third class, i.e. `Authentication`, specifies the authentication mechanism (e.g. pre-shared key).

Figure 6 describes the class `DataProtectionCondition`. In some cases, we need to explicitly select the data to protect. This can be done using explicit conditions ranging on the instances of the `DataSelectionCondition` class. Its subclasses, permit to select particular data types:

- `DBDataSelectionCondition`, that permits to select a portion of the database to protect using the attributes `tableName`, `ViewName` or directly via a SQL query using the `sqlQuery` or the viewName attribute;

- `FileSystemCondition`, that permits to select files, and directories using the `filename` and `path` attributes;

- `XMLDataSelectionCondition`, that includes a consistent set of attributes (`xmlDataType`, `xmlNamespace`, `xmlQueryLanguage`, `xmlQueryLanguageVersion` and `xmlQueryExpression`), to precisely select an entire XML file or a portion. This class is further refined subclassed to `WSSecurityCondition` to introduce more attributes (`headerNameSpace`, `headerLocalName`, `attachment`, `body`, and `content_vs_element`) useful to select data according to the WSSecurity OASIS standard[2].

In other cases, it could be necessary to protect all the data transferred between a service and all its clients, thus we will protect the interfaces. The (abstract) class used to select the interface to which apply data protection is the `InterfaceSelectionCondition`. This class is only used to group all the condition types that can be used to identify interfaces. Interfaces can be selected using the subclasses of the `PacketFilterCondition`, that could be useful to better select the traffic to protect.

[2] `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.pdf`

5 Related Work

Several languages have been developed and proposed during the last ten years. The most significant ones are Policy Core Information Model(PCIM) [15] and eXtensible Access Control Markup Language(XACML) [10].

PCIM provides a useful link with system description and was already used in two EC-funded projects PoSecCo [1] and POSITIF [2] to represent firewall and channel protection configurations. While the XACML is widely adopted for in the specification of ACP (Access Control Policies). In addition to PCIM and XACML, we analysed also ACP languages (SecPAL [5], Usage Control Languages [13], Ponder [8, 9]) and a format to express configurations and policy enforcement points (CIM-SPL [14]).

Ontology-based languages have been considered as well (rei [11], KAoS [18], rein [12, 17]), together with languages represent the other policy-related informations (SAML [6], SCAP [7]).

Unfortunately most of these languages are related to specific context (e.g. access control) or in other cases are complex to be learned and managed by end-users. Therefore, none of the above languages can be considered user-oriented.

6 Conclusion

In this paper, we propose the adoption of two user-oriented security policy languages, namely HSPL and MSPL. The adoption of these languages reduces the complexity of configuring security applications in networks, regardless of the knowledge of the final-user in security and managing security controls. HSPL language is oriented to non-technical user (e.g. customer), while MSPL is designed to be used by administrators and operators. As future work, we plan to add further actions and conditions in the HSPL and improve its usability by exploiting the results of future empirical studies. For what concerns MSPL, we plan to update the policy reconciliation and conflict analysis processes with MSPL model to reduce the number of errors introduced by users during configuration phase.

Acknowledgments

The research described in this paper is part of the SECURED project, co-funded by the European Commission (FP7 grant agreement no. 611458).

References

[1] The PoSecCo project (policy and security configuration management),. `http://www.posecco.eu/` [Online; accessed on May 15, 2018].

[2] The POSITIF project (policy-based security tools and framework),. `http://cordis.europa.eu/project/rcn/75115_en.html` [Online; accessed on May 15, 2018].

[3] C. Basile, D. Canavese, C. Pitscheider, A. Lioy, and F. Valenza. Assessing network authorization policies via reachability analysis. *Computers and Electrical Engineering*, 64(C):110–131, November 2017.

[4] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini. A novel approach for integrating security policy enforcement with dynamic network virtualization. In *Proc. of the 1st IEEE Conference on Network Softwarization (NetSoft'15), London, UK*, pages 1–5. IEEE, April 2015.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, December 2010.

[6] S. Cantor. Security assertion markup language (saml). `https://www.oasis-open.org/standards#samlv2.0` [Online; accessed on May 15, 2018], March 2005.

[7] S. Q. D. Waltermire and K. Scarfone. Specification for the security content automation protocol (scap). Technical Report 800-126, rev.1, NIST SP, February 2011.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems. Technical report, Imperial College Research Report DoC 2000/1, October 2000.

[9] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proc. of the International Workshop on Policies for Distributed Systems and Networks (POLICY'01), Bristol, UK*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer Berlin Heidelberg, February 2001.

[10] S. Godik, A. Anderson, B. Parducci, E. Damiani, P. Samarati, P. Humenn, and S. Vajjhala. eXtensible Access Control Markup Language (XACML) Version 3.0,. Technical report, Organization for the Advancement of Structured Information Standards, January 2013.

[11] L. Kagal. Rei: A policy specification language. Technical report, May 2005.

[12] L. Kagal. The rein policy framework for the semantic web. Technical report, June 2006.

[13] A. Lazouski, F. Martinelli, and P. Mori. Survey: Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, May 2010.

[14] J. Lobo. Cim simplified policy language (cim-spl). Technical report, DMTF Standard, July 2009.

[15] J. S. B. Moore, E. Ellesson, and A. Westerinen. Policy core information model. `https://tools.ietf.org/html/rfc3060` [Online; accessed on May 15, 2018], February 2001. IETF RFC 3060.

[16] J. Strassner. Den-ng: achieving business-driven network management. In *Proc. of the 2002 IEEE/IFIP Network Operations and Management Symposium (NOMS'02), Florence, Italy*, pages 753–766. IEEE, April 2002.

[17] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *Proc. of the International Semantic Web Conference (ISWC'03), Sanibel Island, Florida, USA*, volume 2870 of *Lecture Notes in Computer Science*, pages 419–437. Springer Berlin Heidelberg, October 2003.

[18] A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, July 2004.

[19] F. Valenza, C. Basile, D. Canavese, A. Lioy, F. Valenza, C. Basile, D. Canavese, and A. Lioy. Classification and analysis of communication protection policy anomalies. *IEEE/ACM Transactions on Networking*, 25(5):2601–2614, October 2017.

[20] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy. A formal model of network policy analysis. In *Proc. of the IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI'15), Turin, Italy*, pages 516–522. IEEE, September 2015.

[21] F. Valenza, T. Su, S. Spinoso, A. Lioy, R. Sisto, and M. Vallini. A formal approach for network security policy validation. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 8(1):79–100, March 2017.

Author Biography

Fulvio Valenza received the M.Sc. (summa cum laude) in 2013 and the Ph.D. (summa cum laude) in Computer Engineering in 2017 from the Politecnico di Torino, Torino, Italy. His research activity focus on network security policies. Currently he is a Researcher at the CNR-IEIIT Torino, Italy, where he works on orchestration and management of network security functions in the context of SDN/NFV-based networks and industrial systems.

Antonio Lioy is full professor at the Politecnico di Torino, where he leads the TORSEC research group active in information system security. His research interests include network security, policy-based system protection, and electronic identity. Lioy received a M.Sc. in Electronic Engineering (summa cum laude) and a Ph.D. in Computer Engineering, both from the Politecnico di Torino.